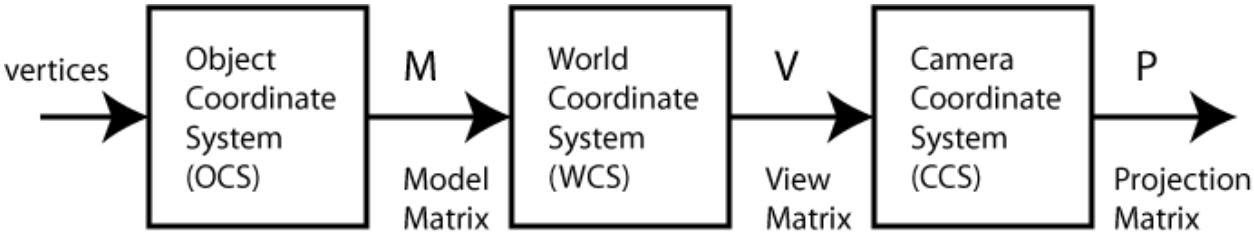
Coordinate Systems

There are 3 coordinate systems that a computer graphics programmer is most concerned with: the Object Coordinate System (OCS), the World Coordinate System (WCS), and the Camera Coordinate System (CCS). Each plays an important role.



OCS: Objects are most easily modeled in a coordinate system with the origin at its center and the coordinate axes aligned in some natural way with the object. For example, the sides of a cube would be aligned with the coordinate axes. Each object will have its own OCS.

WCS: In order for objects to interact, we need to place all objects within a single coordinate system, called the WCS, so their relationship with can be expressed. We think of the WCS as representing the environment in which everything exists. Commonly, the WCS origin establishes the level of the ground and the y axis pointing up. The WCS is typically considered fixed in position over time (objects and camera move within the WCS). The model matrix, M, converts an object's vertices expressed in OCS to the WCS. Each object will have its own M. We use the concept of scene graphs to generate the model matrix M in order to place and move complex objects within the WCS.

CCS: The camera represents the location & orientation of the viewer. Over time, the viewer generally moves around the WCS and so the relationship between the CCS and WCS is not constant. The process of rendering an image requires that a 3D scene (i.e. the vertices) be projected down onto a 2D screen (the image), where the location and orientation of the screen is defined by the camera. The projection is simple to do if the vertices are expressed in the camera coordinates, i.e. with the camera at the origin and looking down the negative z axis (as is the case for the CCS), then the projection will be onto to the xy plane so that the projection is simply $(x,y,z) \rightarrow (x, y)$. Thus, we transform vertices from the WCS to the CCS in order to simplify the projection calculation.

Things to think about:

When specifying the coordinates of a position, it is important to make it clear which coordinates system is being used. Consider the point $(0,0,0)^T$. This will mean something different depending on which coordinate system one is in. For example, in the OCS, the $(0,0,0)^T$ generally corresponds to the center of the object. In the WCS, $(0,0,0)^T$ corresponds to the origin of the world, and in CCS, $(0,0,0)^T$ corresponds to the location of the camera. Similarly, the vector $(1,0,0)^T$ corresponds to the x axis. The x axis in the WCS is not in the same direction as the x axis in the CCS.

In general, a point P can be expressed in any of the coordinate systems. This will be discussed more later.

Transforming OCS →WCS

Consider a cube which might be modeled in the OCS as in Figure 1.

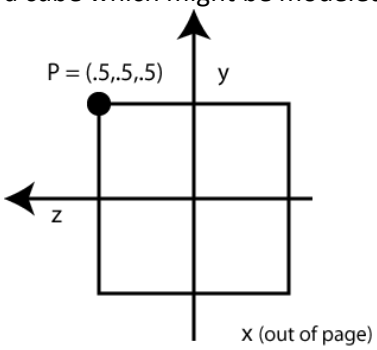


Figure 1: A cube in OCS.

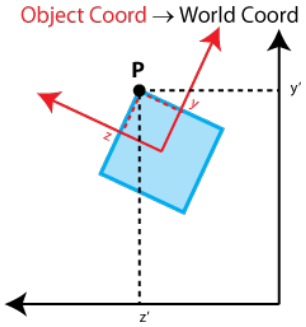


Figure 2: A cube as it appears in OCS and WCS

Let P be a point on a cube such as the vertex at coordinates $(.5,.5,.5)^T$ in OCS. Now suppose we want to place it into "the world" as shown in Figure 2. To do this, we need to transform the coordinates using a matrix we call the *Model Matrix*, **M**.

$$P_{WCS} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = M\, P_{OCS} = M \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Where M is determined using the scene graph techniques.

WCS →CCS

Given the cube in WCS, we can now transform it into the CCS as shown in Figure 3. All three coordinate systems are shown in Figure 4.

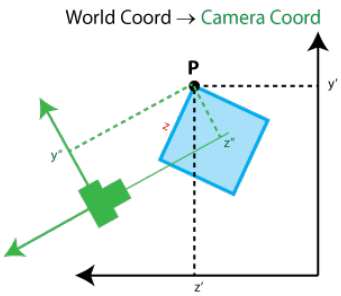


Figure 3: A cube as it appears in WCS and CCS

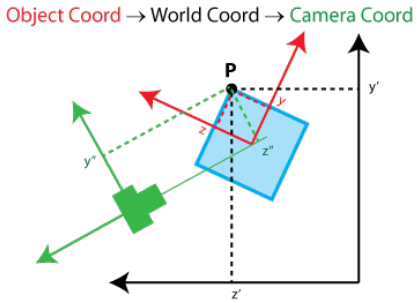


Figure 4: A cube in relation to OCS, WCS, and CCS

Notice that in the CCS the camera always looks down the camera's negative z axis and the y vector points perpendicularly up from the camera. To determine the coordinates in the CCS, we need to transform it from WCS to CCS using a matrix transform which we call the *View Matrix*, **V**.

$$P_{CCS} = \begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} = V\, P_{WCS} = V \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = V\, M\, P_{OCS} = V\, M \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Note: For example, the point $P=(.5,.5,.5)^T$ on the cube as expressed in CCS is

$$P_{CCS} = V\, P_{WCS} = V\, M\, P_{OCS} = V\, M \begin{pmatrix} .5 \\ .5 \\ .5 \end{pmatrix}.$$

Relationship between Transforming Points and Transforming Coordinate Systems

Before discussing how to calculate V, we need to discuss transformations of coordinate systems.

Note that it is not possible to distinguish between translating a point P in a coordinate system versus keeping the point fixed but moving the coordinate axes as shown in Figure 5 below. We present the concepts here in 2D, however, the results extend to 3D.

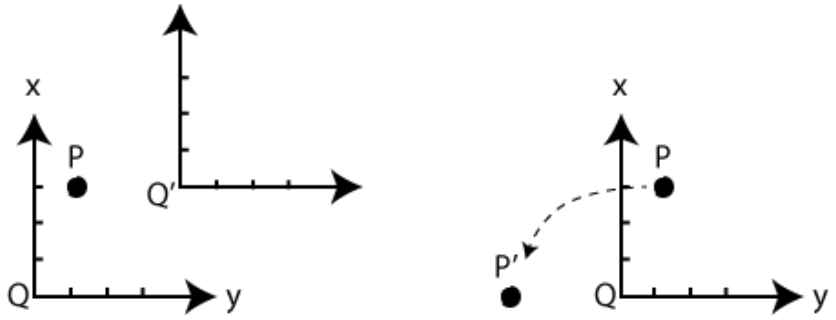


Figure 5: Moving the coordinate system (left) vs moving the point (right)

The relationship between P and Q' (on left) is the same as the relationship between P' and Q (on right):

- On the left, the point P has coordinates (1,3) in Q. We translate the coordinate axes in Q by (4,3) to obtain Q' . Expressed in Q', P now has coordinates (-3,0).
- On the right, we keep Q fixed but translate P=(1,3) by (-4,-3) to that it is now sitting at P'=(-3,0).
- Either way, P ends up at (-3,0) and the transformation that must be applied to the point (on right) is the inverse of that applied to the coordinate system (on left).

In summary, translating a point by a translation T, is equivalent to translating the coordinate system by T^{-1} = inverse of T.

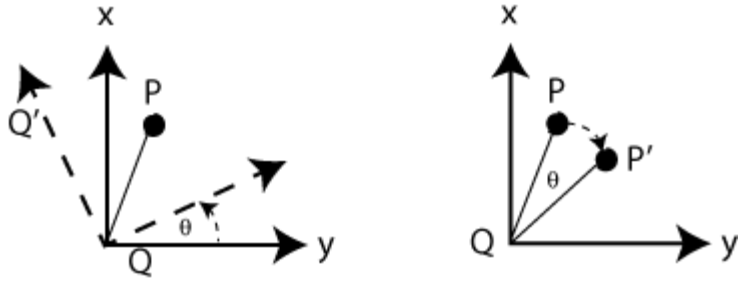


Figure 5: Rotating the coordinate system (left) vs rotating the point (right)

Similarly, it can be shown that *rotating a point by the rotation R, is equivalent to rotating the coordinate system by R^{-1} = inverse of R.*

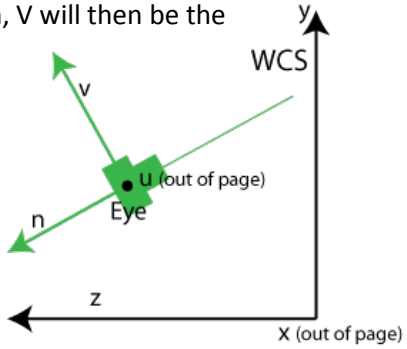
In general, a transformation A applied to the coordinate axes is equivalent to applying the transformation A^{-1} to the vertices.

For example, suppose we apply both a rotation and translation to the coordinate axes: A=TR. This is equivalent to transforming the vertices by $A^{-1} = (TR)^{-1} = R^{-1} T^{-1}$.

We will make use of this result in the next section.

Calculating V

Recall that V is the matrix which transforms vertices in WCS to vertices in CCS. To calculate V, will first consider the transformations necessary to do the equivalent transform of the coordinates axes. Namely, we want to find the matrix transform, TR, consisting of a translation T and rotation R, which will move the WCS to the CCS. Following the result of the previous section, V will then be the inverse of this, namely, $V = R^{-1} T^{-1}$.



Suppose the camera's position in the WCS is given by the *eye* vector

$$\text{eye} = \begin{pmatrix} eye_x \\ eye_y \\ eye_z \end{pmatrix}.$$

The camera's orientation can be defined by the three *orthogonal unit* vectors *u*, *v*, and *n* defined in the WCS, where the camera looks in the direction of *–n*, and where *v* is perpendicularly up and *u* is to the camera's right.

Note: If the WCS and CCS were exactly aligned, then *n* would correspond to the *z* axis, *u* would correspond to the *x* axis, and *v* would correspond to the *y* axis.

The location and orientation of the camera is completely determined by *eye*, *u*, *v*, and *n*. We use these to determine the transforms TR which move the WCS to the CCS. We first apply the rotation, and then the translation.

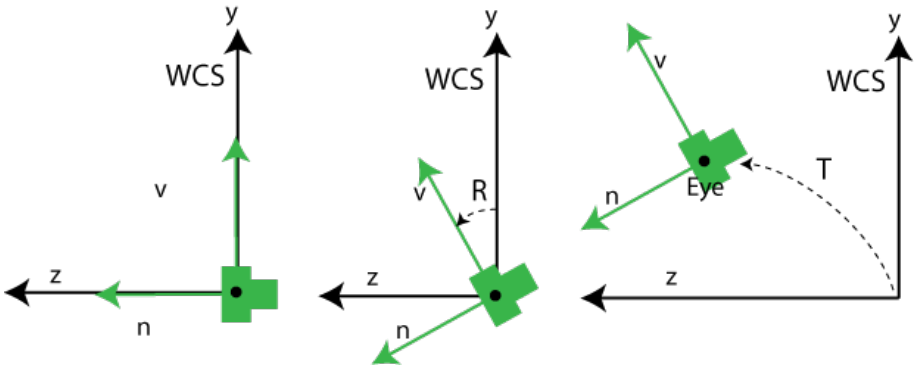


Figure 6: Transforming the axes.

In Figure 6, we begin with the WCS and CCS aligned as on the left. We first rotate by R (middle) and then translate by T (right). It is easy to see that

$$T = \begin{pmatrix} 1 & 0 & 0 & eye_x \\ 0 & 1 & 0 & eye_y \\ 0 & & 1 & eye_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

R is less obvious. Whatever R is, it must satisfy the equations (see middle of Figure 6):

$$u = R\,x, \quad v = R\,y, \quad \text{and} \quad n = R\,z$$

In homogeneous coordinates, we have $u = \begin{pmatrix} u_x \\ u_y \\ u_z \\ 0 \end{pmatrix}$, $v = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix}$, $n = \begin{pmatrix} n_x \\ n_y \\ n_z \\ 0 \end{pmatrix}$, $x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, $y = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$, $z = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$

Note, the 4th component of each is 0 because these are vectors and not points!

So the above equations become

$$u = \begin{pmatrix} u_x \\ u_y \\ u_z \\ 0 \end{pmatrix} = R \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad v = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = R \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \text{ and } n = \begin{pmatrix} n_x \\ n_y \\ n_z \\ 0 \end{pmatrix} = R \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

The only matrix R which satisfies all three of these equations is the matrix whose columns are *u*, *v*, *n* (we are relying on the fact that *u*,*v*, and *n* are *orthonormal* vectors):

$$R = \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Given R and T, we can now calculate V:

$V = R^{-1} T^{-1} = R^T T^{-1} =$

$$\begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z & -u_x eye_x - u_y eye_y - u_z eye_z \\ v_x & v_y & v_z & -v_x eye_x - v_y eye_y - v_z eye_z \\ n_x & n_y & n_z & -n_x eye_x - n_y eye_y - n_z eye_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In the code, we store the position of the *eye* = (eye_x, eye_y ,eye_z,1)

And the view rotation matrix $V_{rot} = R^{-1}$.

Order of Transformations: Things to Think About

We now know how to calculate V and thus how to transform points from OCC to WCS to CCS.

$$P_{CCS} = V\ P_{WCS} = V\ M\ P_{OCS}$$

Think about the following – Do they make sense to you?

$P_{OCS} = (0,0,0,1)^T$ corresponds to the center of the object
$P_{WCS} = (0,0,0,1)^T$ corresponds to the center of the world
$P_{CCS} = (0,0,0,1)^T$ corresponds to the center of the camera

$V\ (0,0,0,1)^T$ gives the location of the world origin in the camera coordinate system
$V\ M\ (0,0,0,1)^T$ gives the location of the object's center in the camera coordinate system
$M\ (0,0,0,1)^T$ gives the location of the object's enter in the world coordinate system

Now suppose we have the standard rotation matrix R_x about the x-axis, that is

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The order in which R_x is applied has a big difference on what happens, as follows:

$V\ M\ R_x\ P_{OCS}$	rotates a vertex P_{OCS} about the object's x axis
$V\ R_x\ M\ P_{OCS}$	rotates a vertex P_{OCS} about the world's x axis
$R_x\ V\ M\ P_{OCS}$	rotates a vertex P_{OCS} about the camera's x axis

Tumble

The tumble transformation changes the view transform V as follows:

- We specify a P_c = "center of interest". This can be anywhere but in the code we limit it to be either the WCS origin or a point a fixed distance in front of the camera.
- Moving the mouse left and right will rotate the camera about the line which goes through P_c and *is parallel to the WCS y-axis*
- Moving the mouse up and down will rotate the camera about the line which goes through P_c and *is parallel to the camera's x axis*

Define the following matrices:

The matrix A rotates along the line which goes through a point P and is parallel to the y-axis

$$A = T(P)\ R_y\ T(-P)$$

where we use the notation T(P) to indicate a translation by an amount P. We want A to be applied in the WCS so it must multiply V on the right. The point P corresponds to the point of interest P_c which is expressed in WCS.

The matrix B rotates along the line which goes through a point P and is parallel to the x-axis.

$$B = T(P)\, R_x\, T(-P)$$

We want B applied in the CCS so we must multiply V by B on the left. The point P corresponds to the point of interest P_c which now must be expressed in CCS, i.e. $P_c' = V_{old}\, P_c$

Thus, to tumble the camera, we have the new view matrix to be

$V_{new} = B\ V_{old}\ A = T(P_c')\, R_x\, T(-P_c')\ \ V_{old}\ \ T(P_c)\, R_y\, T(-P_c)$, where $P_c' = V_{old}\, P_c$

In the code, we do not store V but rather the V_{rot} = rotational part of V, and eye = the camera position. Thus, we need to extract these from V_{new}.

Recall from earlier we had

$V_{new} = V_{rot,new}\ T(-eye_{new})$

$$= \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -eye_{x,new} \\ 0 & 1 & 0 & -eye_{y,new} \\ 0 & 0 & 1 & -eye_{z,new} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{(equation 1)}$$

$$= \begin{pmatrix} u_x & u_y & u_z & -u_x eye_{x,new} - u_y eye_{y,new} - u_z eye_{z,new} \\ v_x & v_y & v_z & -v_x eye_{x,new} - v_y eye_{y,new} - v_z eye_{z,new} \\ n_x & n_y & n_z & -n_x eye_{x,new} - n_y eye_{y,new} - n_z eye_{z,new} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{(equation 2)}$$

Comparing equations 1 and 2 , it is easy to see that

$$V_{rot,new} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

However, it is not so easy to extract eye_{new} because the last column in equation 2 is not eye_{new}. However, if we knew the inverse of $V_{rot,new}$, then we could obtain

$$\begin{aligned} T(-eye_{new}) &= (V_{rot,new}^{-1}\, V_{rot,new})\ \ T(-eye_{new}) \\ &= V_{rot,new}^{-1}\ \ (V_{rot,new}\ T(-eye_{new})) \\ &= V_{rot,new}^{-1}\, V_{new} \end{aligned}$$

Since the inverse of a pure rotation is just its transpose, we can easily obtain $V_{rot,new}^{-1}$, so that we can calculate $T(-eye_{new}) = V_{rot,new}^{-T}\ V_{new}$.

Once we have T(-eye_{new}):

$$T(-eye_{new}) = \begin{pmatrix} 1 & 0 & 0 & -eye_{x,new} \\ 0 & 1 & 0 & -eye_{y,new} \\ 0 & 0 & 1 & -eye_{z,new} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

we see that eye_{new} can be obtained by pulling out the last column of T(-eye_{new}), negating the first three components to obtain eye_{new} = ($eye_{x,new}$, $eye_{y,new}$, $eye_{z,new}$, 1).